A layered model for building ontology translation systems

Oscar Corcho, Asunción Gómez-Pérez

University of Manchester. Kilburn Building, Room 1.17. Oxford Road, Manchester. M13 9PL. United Kingdom

Ontological Engineering Group. Universidad Politécnica de Madrid. 28660 Boadilla del Monte (Madrid), Spain

e-mail: ocorcho@cs.man.ac.uk, asun@fi.upm.es

Editor's notes

To be added

Abstract. We present a model for building ontology translation systems between ontology languages and/or ontology tools, where translation decisions are defined at four different layers: lexical, syntax, semantic, and pragmatic. This layered approach makes ontology translation systems easier to build and understand, and consequently, to maintain and reuse. As part of this model, we propose a method that guides in the process of developing ontology translation systems according to this approach. The method identifies four main activities: feasibility study, analysis of source and target formats, design, and implementation of the translation system, with their decomposition in tasks, and recommends the techniques to be used inside each of them.

1. Introduction

An ontology is defined as a "formal explicit specification of a shared conceptualisation" [Studer et al., 1998], that is, an ontology must be machine readable (it is formal), all its components must be described clearly (it is explicit), it describes an abstract model of a domain (it is a conceptualisation) and it is the product of a consensus (it is shared).

Ontologies can be implemented in varied ontology languages, which are usually divided in two groups: classical and ontology markup languages. Among the classical languages used for ontology construction we can cite (in alphabetical order): CycL [Lenat and Guha, 1990], FLogic [Kifer et al, 1995], KIF [Genesereth and Fikes, 1992], LOOM [MacGregor, 1991], OCML [Motta, 1999], and Ontolingua [Gruber, 1992]. Among the ontology markup languages, used in the context of the Semantic Web, we can cite (also in alphabetical order): DAML+OIL [Horrocks and van Harmelen, 2001], OIL [Horrocks et al, 2000], OWL [Dean and Schreiber, 2004], RDF [Lassila and Swick, 1999], RDF Schema [Brickley and Guha, 2004], SHOE [Luke and Hefflin, 2000], and XOL [Karp et al, 1999]. Each of these languages has its own syntax, its own expressiveness, and its own reasoning capabilities, provided by different inference engines. Languages are also based on different knowledge representation paradigms and combinations of them (frames, first order logic, description logic, semantic networks, topic maps, conceptual graphs, etc.).

A similar situation applies to ontology tools: several ontology editors and ontology management systems can be used to develop ontologies. Among them we can cite (in alphabetical order): KAON [Maedche et al., 2003], OilEd [Bechhofer et al., 2001], OntoEdit [Sure et al., 2002], the Ontolingua Server [Farquhar et al., 1997], OntoSaurus [Swartout et al., 1997], Protégé-2000 [Noy et al., 2000], WebODE [Arpírez et al., 2003], and WebOnto [Domingue, 1998]. As in the case of languages, the knowledge models underlying these tools

have their own expressiveness and reasoning capabilities, since they are also based on different knowledge representation paradigms and combinations of them. Besides, ontology tools usually export ontologies to one or several ontology languages and import ontologies coded in different ontology languages.

There are important connections and implications between the knowledge modelling components used to build an ontology in such languages and tools, and the knowledge representation paradigms used to represent formally such components. With frames and first order logic, the knowledge components commonly used to build ontologies are [Gruber, 1993]: classes, relations, functions, formal axioms, and instances; with description logics, they are usually [Baader et al., 2003]: concepts, roles, and individuals; with semantic networks, they are: nodes and arcs between nodes; etc.

The **ontology translation problem** [Gruber, 1993] appears when we decide to reuse an ontology (or part of an ontology) with a tool or language that is different from those ones where the ontology is available. If we force each ontology-based system developer, individually, to commit to the task of translating and incorporating to their systems the ontologies that they need, they will require both a lot of effort and a lot of time to achieve their objectives [Swartout et al., 1997]. Therefore, ontology reuse in different contexts will be highly boosted as long as we provide ontology translation services among those languages and/or tools.

Many ontology translation systems can be found in the current ontology technology. They are mainly aimed at importing ontologies implemented in a specific ontology language to an ontology tool, or at exporting ontologies modelled with an ontology tool to an ontology language. A smaller number of ontology translation systems are aimed at transforming ontologies between ontology languages or between ontology tools.

Since ontology tools and languages have different expressiveness and reasoning capabilities, **translations between them are not straightforward nor easily reusable**. They normally require to take many **decisions at different levels**, which range from low layers (i.e., how to transform a concept name identifier from one format to the another) to higher layers (i.e., how to transform a ternary relation among concepts to a format that only allows representing binary relations between concepts).

Current ontology translation systems do not usually take into account such a layered structure of translation decisions. Besides, in these systems **translation decisions are usually hidden inside their programming code**. Both aspects make it difficult to understand how ontology translation systems work.

To ameliorate this problem, in this paper we propose a new **model for building and maintaining ontology translation systems**, which identifies four layers where ontology translation decisions can be taken: lexical, syntax, semantic, and pragmatic. This **layered architecture** is based on existing work in formal languages and the theory of signs [Morris, 1938].

This paper is structured as follows: section 2 describes the four layers where ontology translation problems may appear, with examples of how transformations have to be made at each layer. Section 3 describes an ontology translation method based on the previous layers, which is divided into four main activities. Section 4 presents the main conclusions of our work, and section 5 describes related work.

2. Ontology translation layers

As commented above, our ontology translation model proposes to structure translation decisions in four different layers. As commented above, the selection of layers is based on

existing work on formal languages and the theory of signs [Morris, 1938], which consider the existence of several levels in the definition of a language: syntax (related to how the language symbols are structured), semantics (related to the meaning of those structured symbols), and pragmatics (related to the intended meaning of the symbols, that is, how symbols are interpreted or used).

In the context of semantic interoperability, some authors have proposed classifications of the problems to be faced when managing different ontologies in, possibly, different formats. We will enumerate only the ones that are due to differences between the source and target formats[1]. [Euzenat, 2001] distinguishes the following non-strict levels of language interoperability: encoding, lexical, syntactic, semantic, and semiotic. [Chalupsky, 2000] distinguishes two layers: syntax and expressivity (aka semantics). [Klein, 2001] distinguishes four levels: syntax, logical representation, semantics of primitives, and language expressivity, where the last three levels correspond to the semantic layer identified in the other classifications. Figure 1 shows the relationship between these layers.

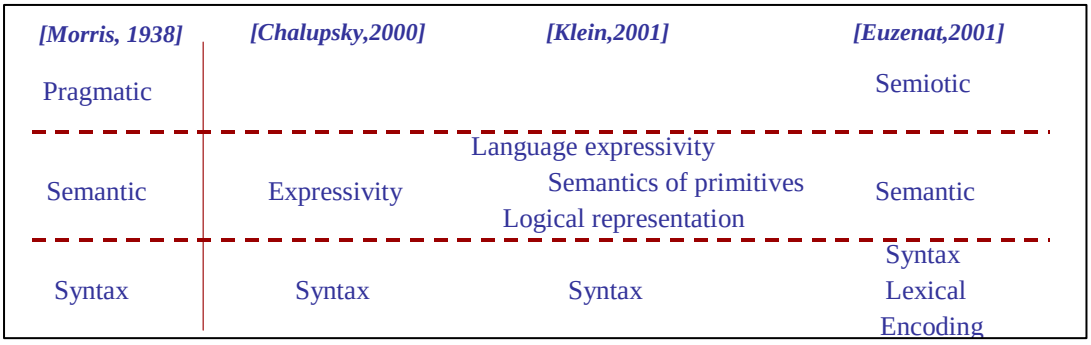

Figure 1. Classifications of semantic interoperability problems and relationships between them.

The layers proposed in our model are mainly based on Euzenat's ones, which is the only one in the context of semantic interoperability who deals with pragmatics (although he uses the term semiotics for it). However, we consider that it is not necessary to split the lexical and encoding layers when dealing with ontologies, and consider them as a unique layer, called lexical.

In the next sections we describe the types of translation problems that can be usually found in each of these layers and will show some examples of common transformations performed in each of them.

2.1 Lexical layer

The lexical layer deals with the "ability to segment the representation in characters and words (or symbols)" [Euzenat, 2001]. Different languages and tools normally use different character sets and grammars for generating their terminal symbols (ontology component identifiers, natural language descriptions of ontology components, and attribute values). This translation layer deals with the problems that may arise in these symbol transformations.

Therefore, in this layer we deal with the following types of transformations:

- **Transformations of ontology component identifiers**. For instance, the source and target formats use different sets of characters for creating identifiers, the source and target format use different naming conventions for their component identifiers, or their

[1] The problems that may appear in the context of semantic interoperability are not only due to the fact that ontologies are available in different formats, but also related to the content of ontologies, their ontological commitments, etc. We only focus on the problems related exclusively to the differences between ontology languages and/or tools.

components have different scopes, and hence some component identifiers cannot overlap with the identifiers of other components.

- **Transformations of pieces of text used for natural language documentation purposes**. For instance, specific characters in the natural language documentation of a component must be escaped since the target format does not allow them as part of the documentation.
- **Transformations of values**. For instance, numbers must be represented as character strings in the target format, or dates must be transformed according to the date formulation rules of the target format.

From a lexical point of view, among the most representative ontology languages and tools we can distinguish three **groups of formats**:

- *ASCII-based formats*. Among these formats we can cite the following classical languages: KIF, Ontolingua, CycL, LOOM, OCML, and FLogic. Also in this group we can include the ontology tools related to some of these languages (the Ontolingua Server, OntoSaurus, and WebOnto). These languages are based on ASCII encodings, and hence the range of characters allowed for creating ontology component identifier, and for representing natural language texts and values is restricted to most of the characters allowed in this encoding.
- *UNICODE-based formats*. Among these formats we can cite the following ontology tools: OntoEdit, Protégé-2000, and WebODE. These formats are based on the UNICODE encoding, which is an extension of the ASCII encoding and thus allows using more varied characters (including Asian and Arabic characters, more punctuation signs, etc.).
- *UNICODE&XML-based formats*. Among these formats we can refer to the ontology markup languages: SHOE, XOL, RDF, RDFS, OIL, DAML+OIL, and OWL, and some of the tools that are related to them, such as KAON and OilEd. These formats are characterized not only for being UNICODE compliant, as the previous ones, but also for restricting the use of some characters and groups of characters in the component identifiers and in the natural language documentation and values, such as the use of tag-style pieces of text (e.g., <*example*>) inside documentation tags. An important restriction is the compulsory use of qualified names (*QNames* as identifiers of ontology concepts and properties, since they are used to construct tags when dealing with instances.

The easiest lexical transformations are usually those to be done from the first and third group of formats to the second one, which is the most unrestricted one. In other cases, the specific features of each format do not allow us to generalize the types of transformations to be done, which mainly consist in replacing non-allowed characters with others that are allowed, or in replacing identifiers that are reserved keywords in a format with other identifiers that are not. Obviously, there are also differences among the languages and tools inside each group, although the transformations needed in those cases are minimal.

Special attention deserves the problem related to the scope of the ontology component identifiers in the source and target formats, and to the restrictions related to overlapping identifiers. These problems appear when, in the source format, a component is defined inside the scope of another and thus its identifier is local to the latter, while the correspondent component has a global scope in the target format. As a consequence, there could be clashes of identifiers if two different components have the same identifier in the source format.

Table 1 shows some examples of how some ontology component identifiers can be transformed from WebODE to Ontolingua, RDF(S), OWL and Protégé-2000, taking into account the rules for generating identifiers in each format and the constraints about the scope and possible overlap of some ontology component identifiers.

As expressed above, inside this layer we also deal with the different naming conventions that exist in different formats[2]. For instance, in Lisp-based languages and tools such as Ontolingua, LOOM, OCML, and their corresponding ontology tools, compound names are usually joined together using hyphens: e.g. `Travel-Agency`. In tools like OntoEdit, Protégé and WebODE, words are separated with blank spaces: e.g. `Travel Agency`. In ontology markup languages, the convention used for class identifiers is to write all the words together, with no blank spaces nor hyphens, and with the first capital letter for each word: e.g. `TravelAgency`.

Table 1. Examples of transformations at the lexical layer.

WebODE identifier	Target	Result	Reasons for transformation
`Business Trip`	Ontolingua	`Business-Trip`	Blank spaces in identifiers are not allowed in Ontolingua
`1StarHotel`	RDF(S)	`OneStarHotel`	Identifiers cannot start with a digit in RDF(S). They do not form valid QNames
Concepts `Name` and `name`	Ontolingua	classes `Name` and `Name_1`	Ontolingua is not case sensitive
Concept `Room` attribute `fare` Concept `Flight` attribute `fare`	OWL	classes `Room`, `Flight` datatypeProperty `roomFare` datatypeProperty `flightFare`	WebODE attributes are local to concepts. OWL datatype properties are not defined in the scope of OWL classes, but globally
Concept `Name` attribute `Name`	Protégé-2000	class `Name`; slot `name`	The identifiers of classes and slots cannot overlap in Protégé-2000

2.2 Syntactic layer

This layer deals with the "ability to structure the representation in structured sentences, formulas or assertions" [Euzenat, 2001]. Ontology components in each language or tool are defined with different grammars. Hence, the syntactic layer deals with the problems related to how the symbols are structured in the source and target formats, taking into account the derivation rules for ontology components in each of them.

In this layer the following types of transformations are included:

- **Transformations of ontology component definitions** according to the grammars of the source and target formats. For instance, the grammar to define a concept in Ontolingua is different than that in OCML.
- **Transformations of datatypes**. For instance, the datatype *date* in WebODE must be transformed to the datatype *&xsd;date* in OWL.

Figure 2 shows an example of how a WebODE concept definition (expressed in XML) is transformed into Ontolingua and OWL. In this example both types of translation problems are dealt with.

Among the most representative ontology languages and tools we can distinguish the following (overlapping) groups of formats:

[2] These types of problems may be also related to the pragmatic layer, as we will describe later in this section. We will also see that the limits of each translation layer are not strict; hence we can find transformation problems that are in the middle of several layers.

- *Lisp-based formats*. The syntax of several classical ontology languages are based on the Lisp language, namely KIF and Ontolingua, LOOM, and OCML, together with their corresponding ontology tools (the Ontolingua Server, OntoSaurus, and WebOnto, respectively).

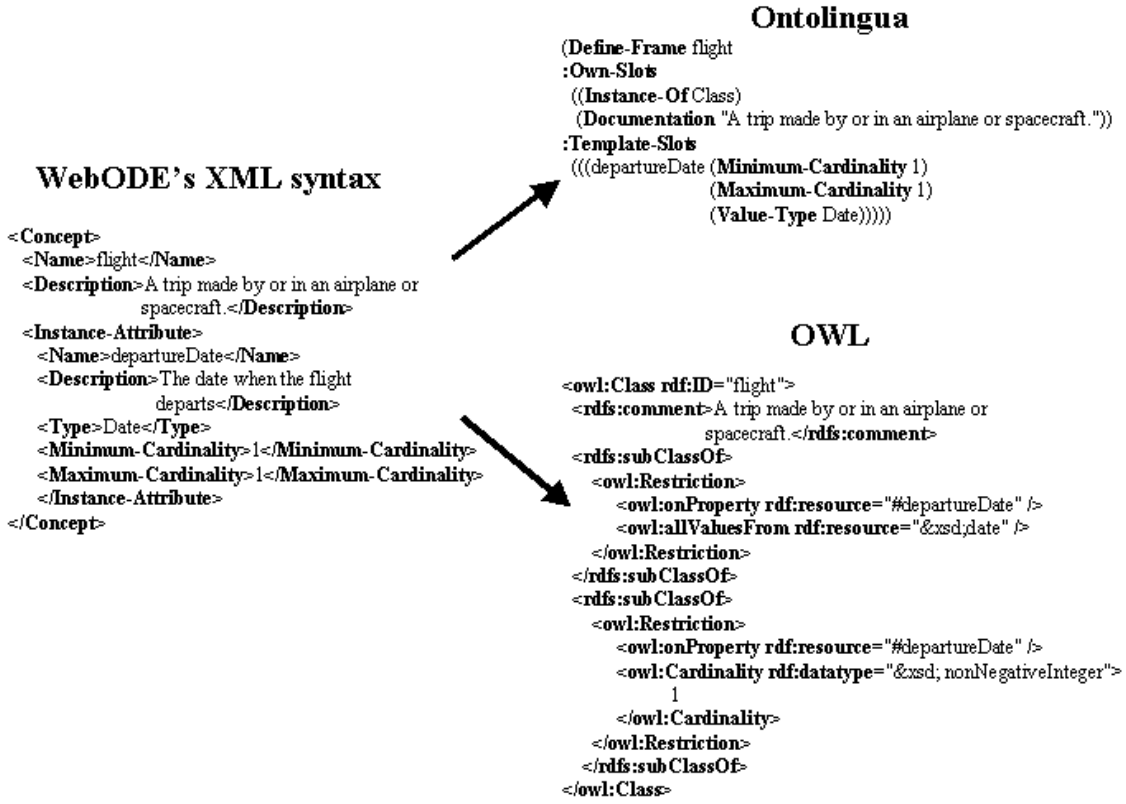

Figure 2. Examples of transformations at the syntactic layer.

- *XML-based formats*. Ontology markup languages are characterized by being represented in XML syntax. Among then we can cite: SHOE, XOL, RDF, RDFS, OIL, DAML+OIL, and OWL. In addition, ontology tools such as OntoEdit, Protégé-2000, and WebODE also provide ad hoc XML backends to implement their ontologies.
- *Ad hoc text formats*. There are other ontology languages that do not provide any of the previous syntaxes but their own ad hoc formats. These languages are F-Logic, the ASCII syntax of OIL, and the Notation-3 (N3) syntax used to represent ontologies in RDF, RDFS, and OWL. Except for F-Logic, these syntaxes are alternative and mainly intended for human-consumption.
- *Ontology management APIs*. Finally, several ontology languages and tools provide ontology management APIs. These APIs are included here because they can be considered as another form of syntax: the expressions used to access, create, and modify ontology components in the programming language in which these APIs are available have to be created according to the specification provided by the API. Among the languages with an ontology management API we have: all the ontology markup languages, where ontologies can be created using available XML Java APIs such as DOM, SAX, etc.; and, more specifically, RDF, RDFS, DAML+OIL, and OWL, for which there are specific APIs that resemble the knowledge models of the ontology languages, such as Jena, the OWL API, etc. Among the tools we have: KAON, OntoEdit, Protégé-2000, and WebODE.

There are other aspects to be considered in this layer, such as the fact that some ontology languages and tools allow defining the same component with different syntaxes. For example, Ontolingua provides at least four different ways to define concepts: using KIF, using the Frame Ontology or using the OKBC-Ontology exclusively, or embedding KIF expressions inside definitions that use the Frame Ontology. This variety adds complexity both for the generation of such a format (we must decide what kind of expression to use[3]) and for its processing (we have to take into account all the possible syntactic variants for the same piece of knowledge).

Inside this layer we must also take into account how the different formats represent datatypes. Two groups can be distinguished:

- *Formats with their own internal datatypes*. Among these formats we can refer to most of the ontology languages, except RDF, RDFS, and OWL, and most of the ontology tools.
- *Formats with XML Schema datatypes*. These datatypes have been defined with the aim of providing datatype standardization in Web contexts (e.g., in Web services). They can be used in the ontology languages RDF, RDFS, and OWL, and in the ontology tool WebODE, which allows using both types of datatypes (internal and XML Schema ones).

Therefore, with regard to datatypes, the problems to be solved will mainly consist in finding the relationships between the internal datatypes of the source and target formats (not all the formats have the same group of datatypes) or finding relationships between the internal datatypes of a format and the XML Schema datatypes, and vice versa.

2.3 Semantic layer

This layer deals with the "ability to construct the propositional meaning of the representation" [Euzenat, 2001]. Different ontology languages and tools can be based on different KR paradigms (frames, semantic networks, first order logic, conceptual graphs, etc.) or on combinations of them. These KR paradigms do not always allow expressing the same type of knowledge, and sometimes the languages and tools based on these KR paradigms allow expressing the same knowledge in different ways.

Therefore, in this layer we deal not only with simple transformations (e.g., WebODE concepts are transformed into Ontolingua and OWL classes), but also with complex transformations of expressions that are usually related to the fact that the source and target formats are based on different KR paradigms (e.g., WebODE disjoint decompositions are transformed into subclass-of relationships and PAL[4] constraints in Protégé-2000, WebODE instance attributes attached to a class are transformed into datatype properties in OWL and unnamed property restrictions for the class).

As an example, figure 3 shows how to represent a concept partition in different ontology languages and tools. In WebODE and LOOM there are specific built-in primitives for representing partitions. In OWL the partition must be represented by defining the *rdfs:subClassOf* relationship between each class in the partition and the parent class, by stating that every possible pair of classes in the decomposition are disjoint, and by defining the parent class as the union of all the classes in the partition. In Protégé-2000 the partition is represented like in OWL, with *subclass-of* relationships between all the classes in the partition and the parent class, with several PAL constraints that represent disjointness between all the classes in the partition, and with the statement that the parent class is abstract (that is, that it cannot have direct instances).

[3] As with naming conventions, this decision will be also related to the pragmatic translation layer.
[4] Protégé Axiom Language

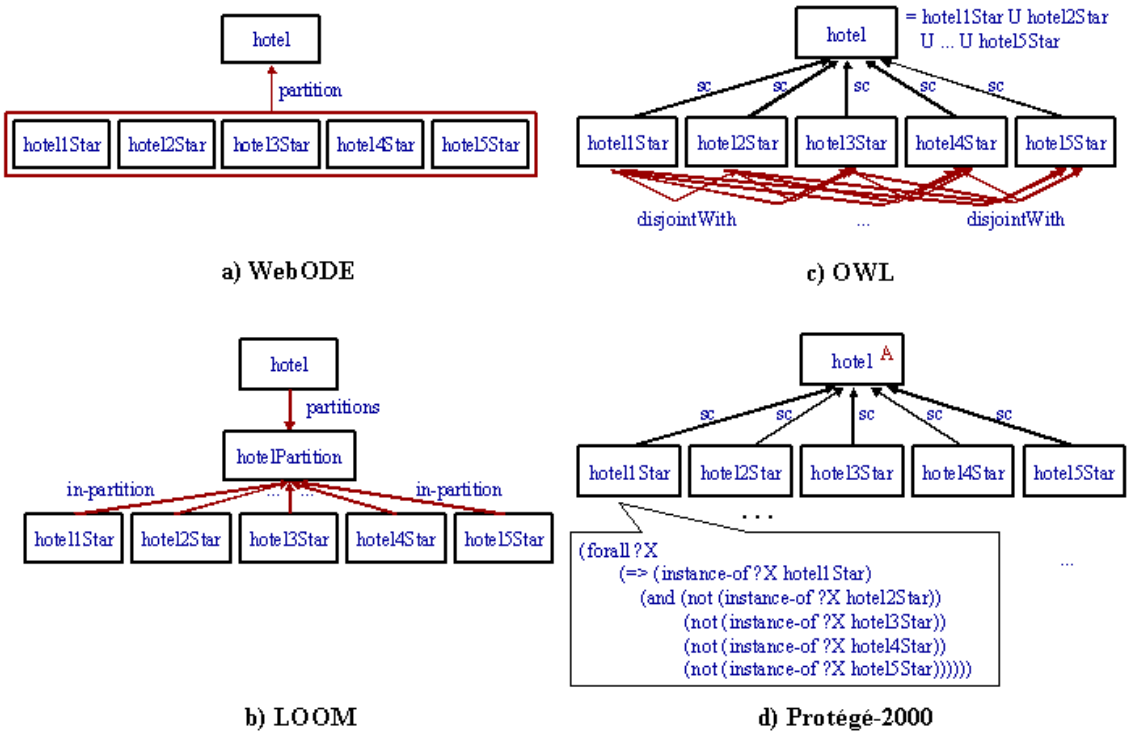

Figure 3. Examples of transformations at the semantic layer.

Most of the work on ontology translation done so far has been devoted to solving the problems that arise in this layer. For example, in the literature we can find several formal, semi-formal, and informal methods for comparing ontology languages and ontology tools' knowledge models ([Baader, 1996], [Borgida, 1996], [Euzenat and Stuckenschmidt, 2003], [Corcho and Gómez-Pérez, 2000], [Knublauch, 2003], etc.), which aim at helping to decide whether two formats have the same expressiveness or not, so that knowledge can be preserved in the transformation. And some of these approaches can be also used to decide whether the reasoning mechanisms present in both formats will allow inferring the same knowledge in the target format.

Basically, these studies allow analysing the expressiveness (and, in some cases, the reasoning mechanisms) of the source and target formats, so that we can know which types of components can be translated directly from a format to another, which types of components can be expressed using other types of components from the target format, which types of components cannot be expressed in the target format, and which types of components can be expressed, although losing part of the knowledge represented in the source format.

Therefore, the catalogue of problems that can be found in this layer are mainly related to the different KR formalisms in which the source and target formats are based. This does not mean that translating between two formats based on the same KR formalism is straightforward, since there might be differences in the types of ontology components that can be represented in each of them. This is specially important in the case of DL languages, since many different combinations of primitives can be used in each language, and hence many possibilities exist in the transformations between them, as shown in [Euzenat and Stuckenschmidt, 2003]. However, the most interesting results appear when the source and target KR formalisms are different.

2.4 Pragmatic layer

This layer deals with the "ability to construct the pragmatic meaning of the representation (or its meaning in context)". Therefore, in this layer we deal with the transformations to be made in the ontology resulting from the lexical, syntactic, and semantic transformations so that both human users and ontology-based applications will notice as less differences as possible with respect to the ontology in the original format, either in one-direction transformations or in cyclic transformations.

Therefore, transformations in this layer will require the following: adding special labels to ontology components so as to preserve their original identifier in the source format; transforming sets of expressions into more legible syntactic constructs in the target format; "hiding" completely or partially some ontology components not defined in the source ontology but that have been created as part of the transformations (such as the anonymous classes commented previously); etc.

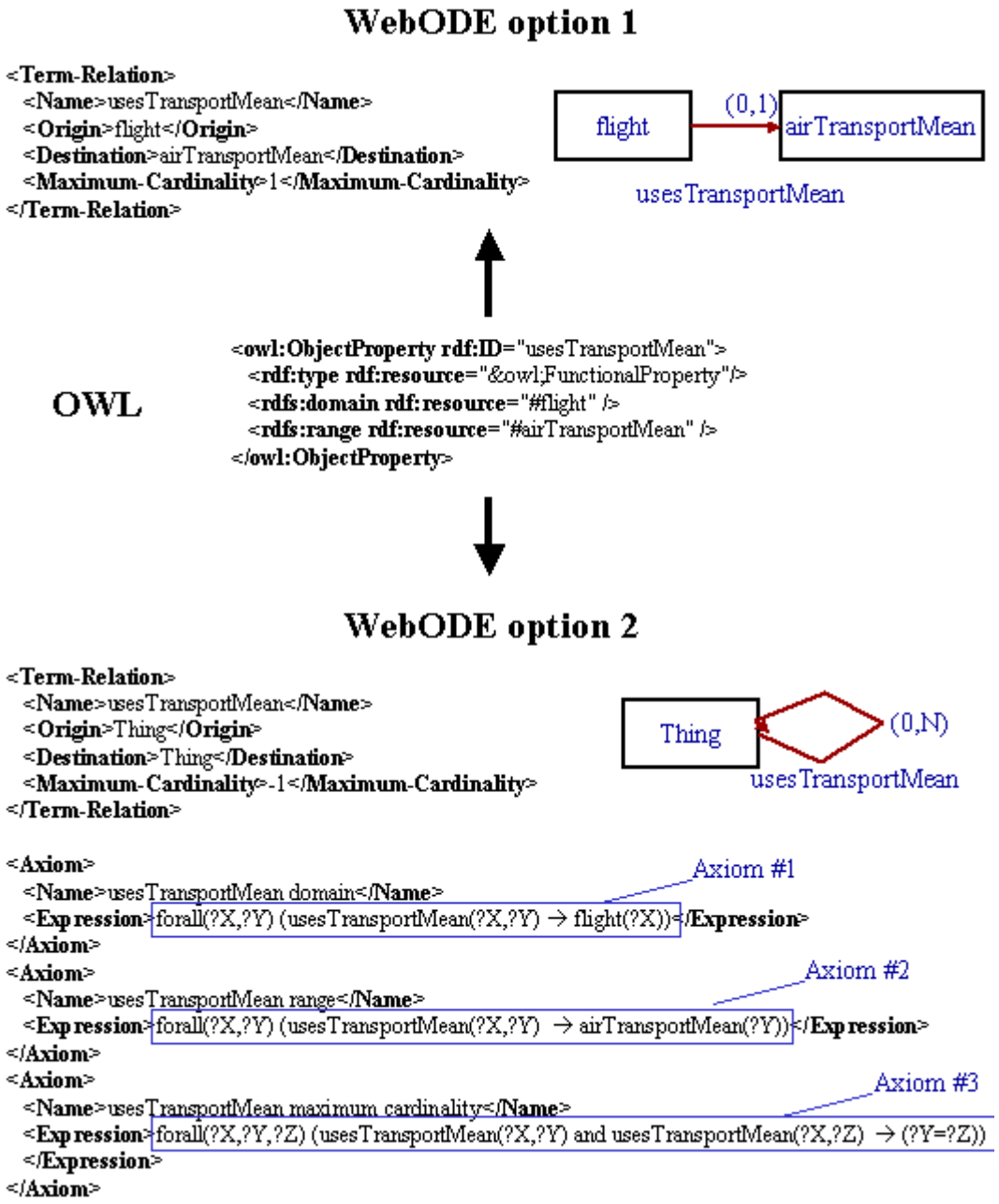

Figure 4. Examples of transformations at the pragmatic layer.

Figure 4 shows two transformations of the OWL functional object property `usesTransportMean` to WebODE. The object property domain is the class `flight` and its range is the class `airTransportMean`. The figure shows two of the possible semantically equivalent sets of expressions that can be obtained when transforming that definition. In the first one, the object property is transformed into the ad-hoc relation `usesTransportMean` that holds between the concepts `flight` and `airTransportMean`, with its maximum cardinality set to one. In the second one, the object property is transformed into the ad-hoc relation `usesTransportMean` whose domain and range is the concept `Thing` (the root of the ontology concept taxonomy), with no restrictions on its cardinality, plus three formal axioms expressed in first order logic: the first one stating that the relation domain is `flight`, the second one that its range is `airTransportMean`, and the third one imposing the maximum cardinality constraint[5].

From a human user's point of view, the first WebODE definition is more legible: at one glance the user can see that the relation `usesTransportMean` is defined between the concepts `flight` and `airTransportMean`, and that its maximum cardinality is one. In the second case, the user must find and interpret the four components (the ad-hoc relation definition and the three formal axioms) to reach the same conclusion.

A similar conclusion can be obtained from an application point of view. Let us suppose that we want to populate the ontology with an annotation tool. The behavior of the annotation tool is different for both definitions. With the first definition, the annotation tool will easily "understand" that its user interface cannot give users the possibility of adding more than one instance of the relation, and that the drop-down lists used for selecting the domain and range of a relation instance will only show direct or indirect instances of the concepts `flight` and `airTransportMean`, respectively. With the second definition, the annotation tool will allow creating more than one relation instance from the same instance and will display all the ontology instances in the drop-down lists, instead of just presenting instances of `flight` and `airTransportMean` respectively. After that, the annotation tool will have to run the consistency checker to detect inconsistencies in the ontology.

2.5 Relationships between ontology translation layers

Figure 5 shows an example of a transformation from the ontology platform WebODE to the language OWL DL. In this example, we have to transform two ad hoc relations with the same name (*usesTransportMean*) and with different domains and ranges (a *flight* uses an *airTransportMean* and a *cityBus* uses a *bus*). In OWL DL the scope of an object property is global to the ontology, and so we cannot define two different object properties with the same name. In this example we show that translation decisions have to be taken at all layers, and we also show how the decision taken at one layer can influence on the decisions to be taken at the others, hence showing the complexity of this task.

[5] We must note that this second option may be obtained because expressions in OWL ontologies may appear in any order in an OWL file, and hence may be processed independently.

Figure 5. Example of translation decisions to be taken at several layers.

Option 1 is driven by semantics: to preserve semantics in the transformation, two different object properties, with different identifiers, are defined. Option 2 is driven by pragmatics: only one object property is defined from both ad hoc relations, since we assume that they refer to the same meaning, but some knowledge is lost in the transformation (the one related to the object property domain and range). Finally, option 3 is also driven by pragmatics, with more care on the semantics: again, only one object property is defined, and its domain and range is more restricted than in option 2, although we still lose the exact correspondence between each domain and range.

3. A layered ontology translation method

Once that we have described the four layers where ontology translation decisions have to be taken, we will present our method for building ontology translation systems, based on these layers. This method consists of four activities: feasibility study, analysis of source and target formats, design and implementation of the translation system. As we will describe later, these activities are divided into tasks, which can be performed by different sets of people and with different techniques.

Ontology translation systems are difficult to create, since many different types of problems have to be dealt with in them. Consequently, this method recommends to develop ontology translation systems following an iterative life cycle. It proposes to identify a first set of expressions that can be easily translated from one format to another, so that the first version of the ontology translation system can be quickly developed and tested; then it proposes to refine the transformations performed, to analyse more complex expressions and to design and implement their transformations, and so on and so forth. The reason for such a recommendation is that developing an ontology translation system is usually a complex task that requires taking into account too many aspects of the source and target formats, and many different types of decisions on how to perform specific translations. In this sense, an iterative life cycle ensures that complex translation problems are tackled once that the developers have a better knowledge of the source and target formats and once that they have tested simpler translations performed with earlier versions of the software produced.

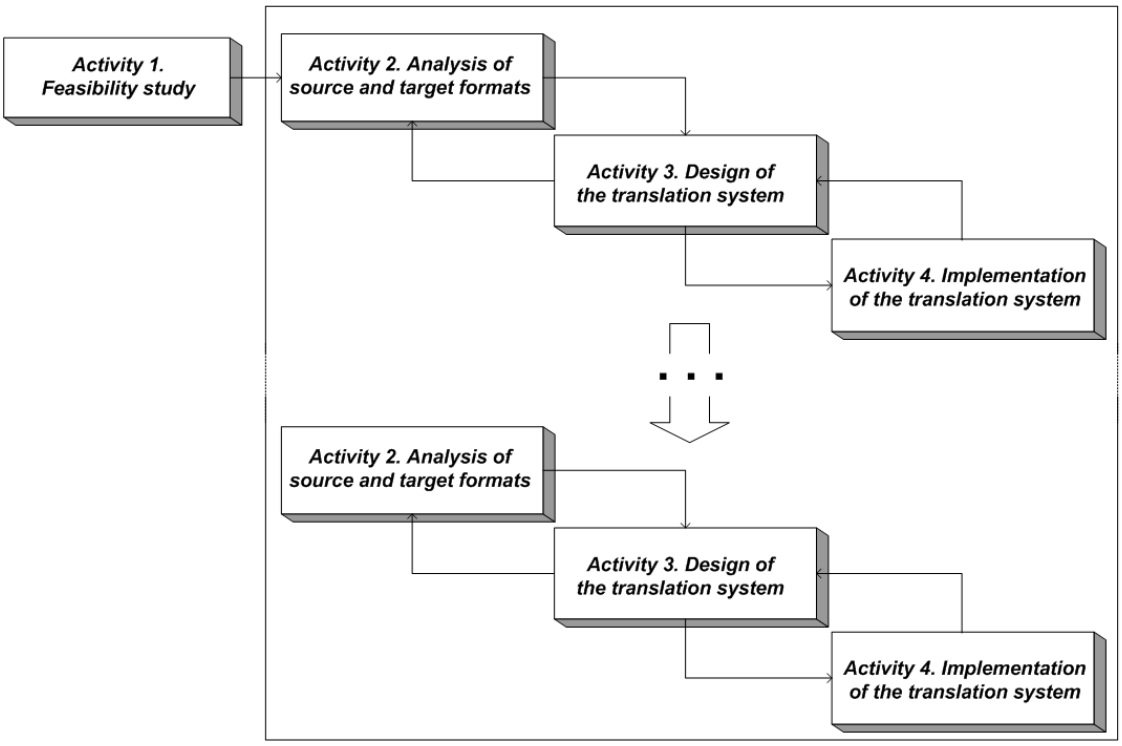

Figure 6. Proposed development process of ontology translation system.

The feasibility activity is performed at the beginning of the development project. If this study recommends to start with the ontology translation system development, then for each cycle, the other three activities will be performed sequentially, although developers can always go back to a previous activity using the feedback provided by the subsequent ones, as shown in figure 6, which summarises the proposed development process.

As a summary, table 2 lists the activities that the method proposes and the tasks to be performed inside each activity. The design and implementation activities take into account the four translation layers described in the previous section.

Table 2. List of activities and tasks of the method for developing ontology translation systems.

Activity	***Task***
1. Feasibility study	1.1. Identify ontology translation system scope 1.2. Analysis of current ontology translation systems 1.3. Ontology translation system requirement definition 1.4. Feasibility decision-making and recommendation
2. Analysis of source and target formats	2.1. Describe source and target formats 2.2. Determine expressiveness of source and target formats 2.3. Compare knowledge models of source and target formats 2.4. Describe and compare additional features of source and target formats 2.5. Determine the scope of translation decisions 2.6. Specify test plan
3. Design of the translation system	3.1. Find and reuse similar translation systems 3.2. Propose transformations at the pragmatic level 3.3. Propose transformations at the semantic level 3.4. Propose transformations at the syntax level 3.5. Propose transformations at the lexical level 3.6. Propose additional transformations
4. Implementation of the translation system	4.1. Find translation functions to be reused 4.2. Implement transformations in the pragmatic level 4.3. Implement transformations in the semantic level 4.4. Implement transformations in the syntax level 4.5. Implement transformations in the lexical level 4.6. Implement additional transformations 4.7. Declarative specification processing and integration 4.8. Test suite execution

The method does not put special emphasis in other activities that are usually related to software systems' development, either specific to the software development process, such as deployment and maintenance, or related to support activities, such as quality assurance, project management and configuration management. Nor does it emphasise other tasks usually performed during the feasibility study, analysis, design, and implementation activities of general software systems' development. It only describes those tasks that are specifically related to the development of ontology translation systems, and recommends to perform such additional activities and tasks, which will be beneficial to their development.

In the following sections we will describe briefly the objective of each of these activities, the techniques that can be used to perform them, and their inputs and outputs.

3.1 Feasibility study

The objective of this activity is to analyse the ontology translation needs, so that the proposed solution takes into account not only the technical restrictions (technical feasibility), but also other restrictions related to the business objectives of an organization (business feasibility) and to the project actions that can be successfully undertaken (project feasibility). As a result of this activity, the main requisites to be satisfied by the ontology translation system are obtained, and the main costs, benefits, and risks are identified. The most important aspect of this feasibility study regards the technical restrictions, which can determine whether it is recommended or not to proceed with the ontology translation system development.

The techniques (and documents) used in the execution of these tasks are inspired by knowledge engineering approaches (such as [Gómez-Pérez et al., 1997] and [Schreiber et al., 1999]), and mainly based on the CommonKADS worksheets.

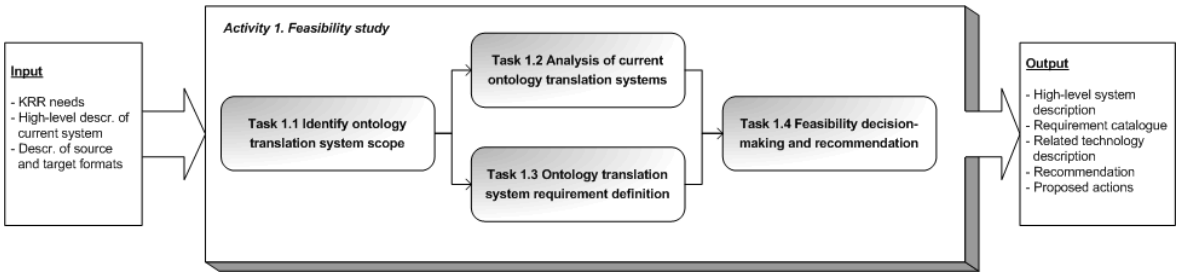

Figure 7. Task decomposition of activity 1 (feasibility study).

As shown in figure 7, we first propose to determine the scope of the ontology translation system that will be implemented, its expected outcome, the context where it will be used, etc. We then propose to analyse current translation systems available between the source and target formats, and determine the requisites of the new system. Finally, we propose to fill in a checklist where the three dimensions identified above are considered (technical, business, and project feasibility), allowing us to make a decision on the feasibility of the system and to propose a set of actions and recommendations to be followed.

Consequently, the input to this activity consists in some preliminary high-level information about current systems, the KRR needs and the source and target formats. The results consist in a deeper description of the current ontology translation systems available for the origin and target formats, a preliminary catalogue of requisites for the system to be developed and the recommendation about its feasibility, including the main costs, benefits, and risks involved.

3.2 Analysis of the source and target formats

The objective of this activity is to obtain a thorough description and comparison of the source and target formats of the ontology translation system. We assume that they will allow us to gain a better understanding of their similarities and differences in expressiveness, which will be useful to design and implement the translation decisions in the subsequent activities. Besides, in this activity we refine the catalogue of requirements already obtained as a result of the feasibility study, and we identify the test suite that will be used to test the translation system validity after each iteration in its development process. A summary of the tasks to be performed, and of the input and outputs of this activity, is shown in figure 8.

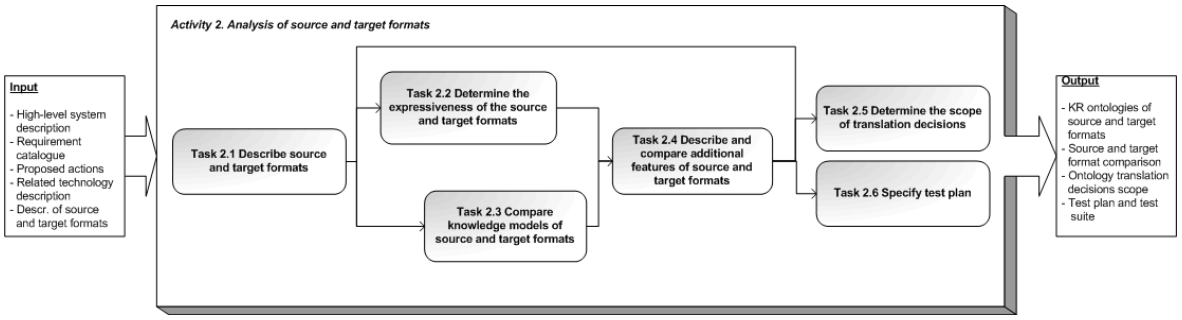

Figure 8. Task decomposition of activity 2 (analysis of source and target formats).

Many techniques can be used to describe the source and target formats of the translation system. Among them, the method recommends describing their KR ontologies, as shown

in [Broekstra et al., 2000] or [Gómez-Pérez et al., 2003], which provide a good overview of the ontology components that can be used to represent ontologies with them.

For the comparison tasks we can use either formal, semi-formal or informal approaches, such as the ones identified in section 2.3 (which show good examples of the results that should be obtained). Once that the two formats have been described, evaluated and compared, we recommend to focus on other additional features that could be needed in the translation process. They may include reasoning mechanisms or any other specific details that could be interesting for the task of translation.

The information gathered in the previous tasks is used to determine the scope of the translation decisions to be made: which components map to each other, which components of the source format must be represented by means of others in the target format, which components cannot be represented in the target format, etc. As a result, we obtain a refinement of the requirement catalogue obtained during the feasibility study, which serves as the basis for the next activities (design and implementation of the translation system).

Finally, we propose to define the test plan, which consists of a set of unitary tests that the translation system must pass in order to be considered valid. The test suite must consider all the possible translation situations that the translation system must cover. These ontologies will be available in the source format and in the target format (which should be the output of the translation process). The test execution will consist on comparing the output obtained and the output expected. For each iteration of the software development process we will define different sets of ontologies.

This activity receives as an input all the results of the feasibility study, together with the description of the source and target formats (also used as an input for that activity). It outputs a comparison of both formats, the scope of the translation decisions to be performed, with a refined requirements catalogue, and a test plan with its corresponding test suite.

3.3 Design of the ontology translation system

The design activity aims at providing a detailed specification of the transformations to be performed by the ontology translation system. From this specification we will be able to generate the implementation of the translation decisions at each layer, which will be used in its turn to generate the final ontology translation system. The tasks, inputs and outputs of this activity are shown in figure 9.

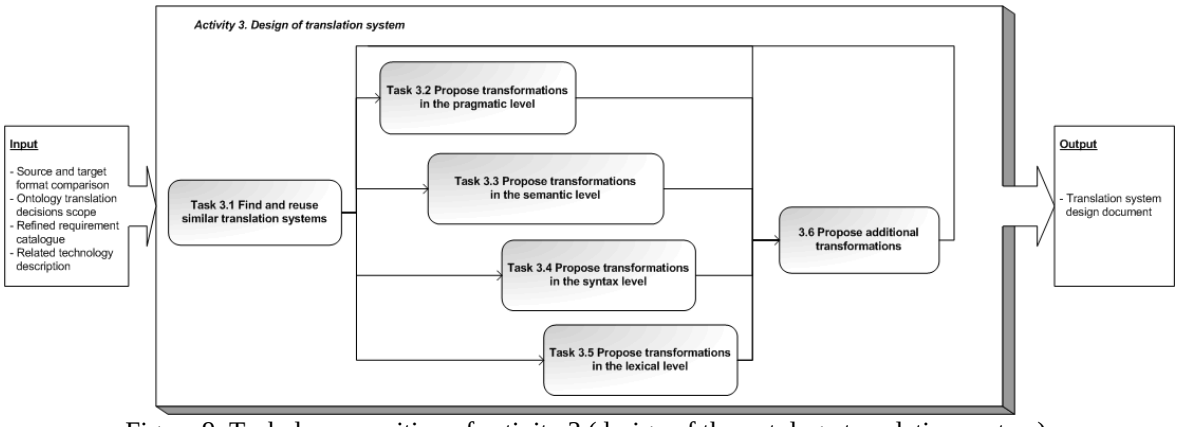

Figure 9. Task decomposition of activity 3 (design of the ontology translation system).

The objective of the first task is to analyse similar ontology translation systems and to detect which of their translation decisions can be actually reused. We assume that by reusing existing translation decisions we will be able to minimise the sources of errors in our translation proposals. Furthermore, we will benefit from work already known, for

which we already know its properties (namely, how they preserve semantics and pragmatics). We must remember that the potential reusable systems were already identified and catalogued during the feasibility study.

The second group of tasks deals with the four layers of translation problems described in section 2: we propose to design transformations at the different inter-related levels, using different techniques for each layer. All these tasks should be mainly performed in parallel, and the decisions taken at one task provide feedback for the others, as shown in the figure. We propose to start with the translation decisions at the pragmatic and semantic levels, leaving the syntax and lexical transformations for the last steps. The pragmatic and semantic translation decisions are mainly proposed by knowledge engineers, while the syntax and lexical transformations can be proposed jointly by knowledge and software engineers, since they have more to do with general programming aspects, rather than with the complexity of transforming knowledge. The method proposes to represent these translation decision mainly with tables and diagrams, such as the ones proposed in table 3 and figure 10 for transformations between WebODE to OWL DL.

Table 3. Semantic transformation of WebODE partitions to OWL DL.

WebODE	**OWL DL**
Partition (C, {C1,C2,…,Cn})	$C \equiv C1 \cup C2 \cup \ldots \cup Cn$ $Ci \subseteq C \quad \forall Ci \in \{C1,C2,\ldots,Cn\}$ $Ci \cap Cj \subseteq \perp \quad \forall Ci \neq Cj,, Ci,Cj \in \{C1,C2,\ldots,Cn\}$

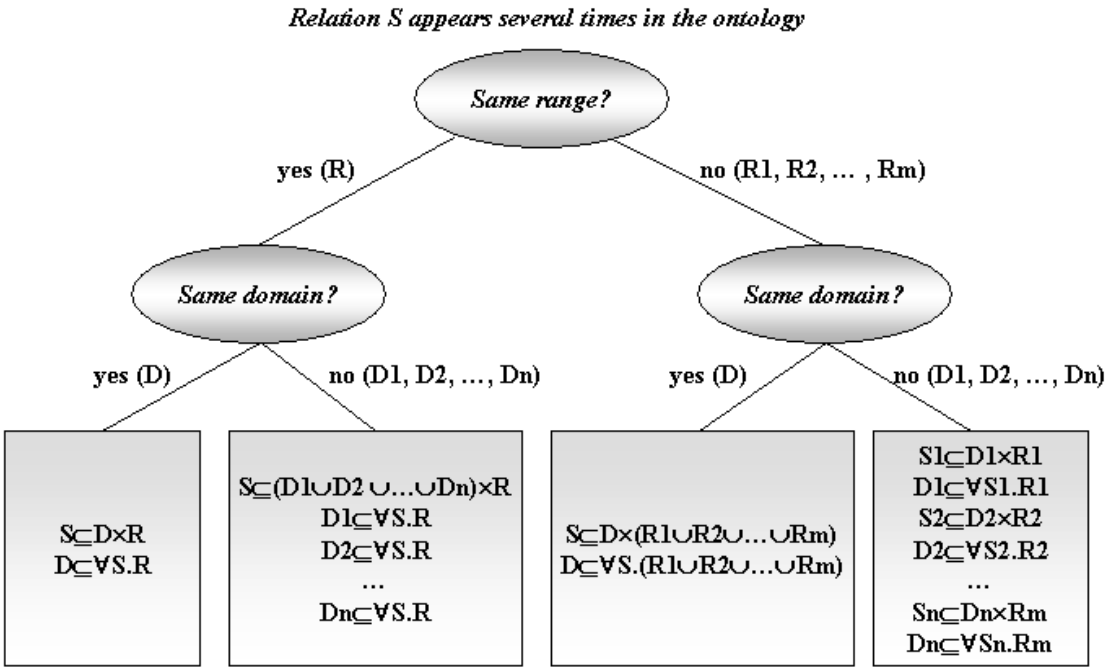

Figure 10. Pragmatic transformations with regard to the scope of WebODE ad hoc relations

Finally, the objective of the last task is to propose any additional transformations or design issues that have not been covered by the previous tasks, because they could not be catalogued as lexical, syntax, semantic, or pragmatic transformations, which are necessary for the correct functioning of the ontology translation system. These transformations include design issues such as the initialisation and setting up of parameters in the source and target formats, any foreseen integration needs of the generated system in the case of transformations where ontology tools or specific libraries are used, etc.

As shown in figure 9, we may need to come back to the second group of activities after proposing some additional transformations. This is a cyclic process until we have

determined all the transformation to be performed in the corresponding development iteration. All the output results obtained from the tasks in this activity are integrated in a single document called “translation system design document”, as shown in the figure.

3.4 Implementation of the translation system

The objective of the implementation activity is to create the declarative specifications of the transformations to be performed by the ontology translation system, which will be used to generate its final code. The method proposes to implement these translations using three different formal languages: ODELex, ODESyntax, and ODESem, which correspond to the lexical, syntax, and semantic/pragmatic ontology translation layers, respectively. The same language (ODESem) is used for implementing semantic and pragmatic transformations because the translation decisions at both layers are similar. The description of these languages is out of the scope of this paper and can be found in [Corcho and Gómez-Pérez, 2004] and [Corcho, 2005]. We can just say that the ODELex and ODESyntax languages are similar to the lex [Lesk, 1975] and yacc [Johnson, 1975] languages, used for compiler construction, and that ODESem is based on common rule-based systems.

Like in the design activity, the tasks inside this implementation activity are divided in groups, which are four in this case, as shown in figure 11.

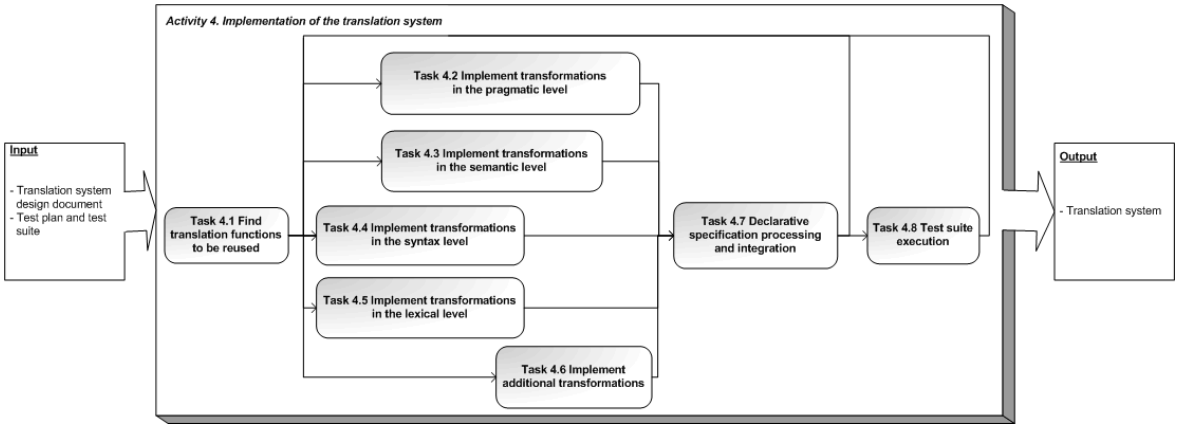

Figure 11. Task decomposition of activity 4 (implementation of the ontology translation system).

The goal of the first task is to select reusable pieces of code from the declarative specifications of other ontology translation systems. These pieces of code are selected on the basis of the results obtained from the first task of the design activity, and can be related to any of the four translation layers.

The next five tasks are grouped together and should be performed almost in parallel, as shown in the figure. In these tasks software and knowledge engineers must actually implement the transformations at the four layers: lexical, syntax, semantic, and pragmatic, and the additional transformations described in task 3.6. Unlike in the design activity, we propose to start with the low-level transformations (those at the lexical and syntax layers), and continue with the more abstract (and difficult) ones. The reason for the task ordering suggested is that the semantic and pragmatic transformation implementations usually need to take into account the specific implementations at the lexical and syntax layers. We are currently developing automatic tools that transform the declarative specifications in ODELex, ODESyntax and ODESem into Java code.

In the next task (4.7: declarative specification processing and integration) the software engineer is in charge of transforming the previous declarative implementations at all levels, plus the additional transformations, into actual running code, which will perform the translations as specified in the previous code. Besides, the software engineer has to integrate the resulting ontology translation system into another information system (such

as an ontology tool), if required. Given that most of the transformations have been implemented in formal languages, most of the processes involved in this task can be automated. If problems are detected during this task, the method recommends to go back to the implementation activities so as to solve them.

Finally, the method proposes to execute the test suite that was defined during the analysis activity, which are considered as the system tests for our system. This does not prevent us from defining and executing other kinds of tests (from unitary tests to integration tests) at any point during the development. This task consists on inputting the ontologies in the test suite to the resulting ontology translation system and checking whether the output corresponds to the one expected. Note that in most of the cases, this check will consist in comparing whether the output file(s) and the expected file(s) are identical, but there are cases where this kind of comparison will not be possible, since the results can come in any order (for instance, in RDF and OWL ontologies). If any of the test fails, we must go back to the previous implementation activities to detect the problems. Furthermore, we must consider that the method allows moving to previous activities if problems are detected at any point of our development.

4. Conclusions

This paper presents two important contributions to the current state of the art on ontology translation. First, it proposes to consider that ontology translation problems can appear at four different layers, which are inter-related, and describe the most common problems that may appear at each of those layers. Some existing approaches have identified similar layers in ontology translation. However, these and other approaches have mainly focused on the problems related to the semantic layer, and have not considered the other ones, which are also important for building systems that make "good quality" translations. The low quality of some translation systems has been recently shown in the interoperability experiment performed for the ISWC2003 workshop on Evaluation of Ontology Tools[6]. The results obtained in this workshop showed that making good translation decisions at the lexical, syntax and pragmatic levels is also as important as making good translation decisions at the semantic level.

The second main contribution of this paper is related to the fact that it is the first approach that gives an integrated support for the complex task of building ontology translation systems. As we commented in the introduction, ontology translation systems are not easy to create, and are difficult to maintain as well. Most of the translation systems currently available have been developed ad hoc, the translation decisions that they implement are usually difficult to understand and hidden in the source code of the systems, and, besides, it is not clear nor documented how much knowledge is lost in the transformations that they perform. There are many complex decisions that have to be implemented, and these decisions are usually taken at the low implementation level, instead of performing a detailed analysis and design of the different translation choices available and taking a decision based on the actual ontology translation requirements. The method proposed in this paper helps in this task by identifying clearly the activities to be performed, the tasks in which each activity is decomposed, and how these tasks have to be performed, the inputs and outputs of the activities, and the set of techniques that can be used to perform them. Besides, a set of declarative languages are proposed, although not described in this paper, to help in the implementation of translation decisions.

This method has been derived from our long experience in the generation of ontology translation systems from the ontology engineering platform WebODE to different ontology languages and tools, and viceversa (12 systems), and has been used for building other six

ontology translation systems. These systems have been succesfully built by different people with background on knowledge and software engineering, following the method proposed in this paper, and the techniques identified for each task.

6. Related work

Though there are no other integrated methods for building ontology translation systems available, we can find some technology that allows creating them. Specifically, we can cite two tools: Transmorpher and OntoMorph:

- *Transmorpher*[7] [Euzenat and Tardif, 2001] is a tool that facilitates the definition and processing of complex transformations of XML documents. Among other domains, this tool has been used in the context of ontologies, using a set of XSLT documents that are able to transform from one DL language to another, expressed in DLML[8]. This tool is aimed at supporting the "family of ontology languages" approach for ontology translation, described in [Euzenat and Stuckenschmidt, 2003]. The main limitation of this approach is that it only deals with problems in the semantic layer, not focusing on other problems related to the lexical, syntax and pragmatic layers.
- *OntoMorph* [Chalupsky, 2000] is a tool that allows creating translators declaratively. Transformations between the source and the target formats are specified by means of pattern-based transformation rules, and are performed in two phases: syntactic rewriting and semantic rewriting. The last one needs the ontology or part of it translated into PowerLoom, so that this KR system can be used for certain kinds of reasoning, such as discovering whether a class is subclass of another, whether a relation can be applied to a concept or not, etc. Since this tool is based on PowerLoom (and consequently on Lisp), it cannot handle easily all the problems that may appear in the lexical and syntax layers.

Although these tools do not give an integrated support for the task of building ontology translation systems, this does not mean that they cannot be used as a technological support for the method proposed in this paper, especially for the implementation activity.

Acknowledgements

This work has been supported by by the IST project Esperonto (IST-2001-34373). Part of this paper is based on section 3.4 of the book "A Layered Declarative Approach to Ontology Translation with Knowledge Preservation", published by IOS Press.

References

[1] Arpírez JC, Corcho O, Fernández-López M, Gómez-Pérez A (2003) *WebODE in a nutshell*. AI Magazine 24(3):37-48. Fall 2003

[2] Baader F (1996) *A Formal Definition for the Expressive Power of Terminological Knowledge Representation Languages*. Journal of Logic and Computation 6(1):33–54

[3] Baader F, McGuinness D, Nardi D, Patel-Schneider P (2003) *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, Cambridge, United Kingdom

[4] Bechhofer S, Horrocks I, Goble C, Stevens R (2001) *OilEd: a reasonable ontology editor for the Semantic Web*. In: Baader F, Brewka G, Eiter T (eds) Joint German/Austrian conference on Artificial Intelligence (KI'01). Vienna, Austria. (Lecture Notes in Artificial Intelligence LNAI 2174) Springer-Verlag, Berlin, Germany, pp 396–408

[5] Borgida A (1996) *On the relative expressiveness of description logics and predicate logics*. Artificial Intelligence 82(1-2):353–367

[6] Brickley D, Guha RV (2004) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. *http://www.w3.org/TR/PR-rdf-schema*

[7] Chalupsky H (2000) *OntoMorph: a translation system for symbolic knowledge*. In: Cohn AG, Giunchiglia F, Selman B (eds) 7th International Conference on Knowledge Representation and Reasoning (KR'00). Breckenridge, Colorado. Morgan Kaufmann Publishers, San Francisco, California, pp 471–482

[8] Corcho O (2005) *A layered declarative approach to ontology translation with knowledge preservation*. Frontiers in Artificial Intelligence and its Applications. Dissertations in Artificial Intelligence. IOS Press

[9] Corcho O, Gómez-Pérez A (2000) *A Roadmap to Ontology Specification Languages*. In: Dieng R, Corby O (eds) 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00). Juan-Les-Pins, France. Springer-Verlag, Lecture Notes in Artificial Intelligence (LNAI) 1937, Berlin, Germany, pp 80–96

[10] Corcho O, Gómez-Pérez A (2004) *ODEDialect: a set of declarative languages for implementing ontology translation systems*. In: Koubarakis M, Goble C, Gómez-Pérez A, Euzenat J (eds) ECAI2004 Workshop on Semantic Intelligent Middleware for the Web and the Grid, Valencia, Spain

[11] Dean M, Schreiber G (2004) *OWL Web Ontology Language Reference*. W3C Recommendation. *http://www.w3.org/TR/owl-ref/*

[12] Domingue J (1998) *Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web*. In: Gaines BR, Musen MA (eds) 11th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'98). Banff, Canada, KM4:1–20

[13] Euzenat J (2001) *Towards a principled approach to semantic interoperability*. In: Gómez-Pérez A, Grüninger M, Stuckenschmidt H, Uschold M (eds) IJCAI2001 Workshop on Ontologies and Information Sharing, Seattle, Washington

[14] Euzenat J, Stuckenschmidt H (2003) *The `family of languages' approach to semantic interoperability*. In: Omelayenko B, Klein M (eds) Knowledge transformation for the semantic web, IOS press, Amsterdam, The Netherlands, pp49-63

[15] Euzenat J, Tardif L (2001) *XML transformation flow processing*. Markup languages: theory and practice 3(3):285–311

[16] Farquhar A, Fikes R, Rice J (1997) *The Ontolingua Server: A Tool for Collaborative Ontology Construction*. International Journal of Human Computer Studies. 46(6):707–727

[17] Genesereth MR, Fikes RE (1992) *Knowledge Interchange Format. Version 3.0. Reference Manual*. Technical Report Logic-92-1. Computer Science Department. Stanford University, California, available at *http://meta2.stanford.edu/kif/Hypertext/kif-manual.html*

[18] Gómez-Pérez A, Fernández-López M, Corcho O (2003) *Ontological Engineering: with examples from the areas of knowledge management, e-commerce and the Semantic Web*, Springer-Verlag, New York.

[19] Gómez-Pérez A, Juristo N, Montes C, Pazos J (1997) *Ingeniería del Conocimiento*. Centro de Estudios Ramón Areces

[20] Gruber TR (1992) *Ontolingua: A Mechanism to Support Portable Ontologies*. Technical report KSL-91-66, Knowledge Systems Laboratory, Stanford University, Stanford, California. *ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-91-66.ps*

[21] Gruber TR (1993) *A translation approach to portable ontology specification*. Knowledge Acquisition 5(2):199–220

[22] Horrocks I, Fensel D, Harmelen F, Decker S, Erdmann M, Klein M (2000) *OIL in a Nutshell*. In: Dieng R, Corby O (eds) 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00). Juan-Les-Pins, France. Springer-Verlag, Lecture Notes in Artificial Intelligence (LNAI) 1937, Berlin, Germany, pp 1–16

[23] Horrocks I, van Harmelen F (eds) (2001) *Reference Description of the DAML+OIL (March 2001) Ontology Markup Language*. Technical report. *http://www.daml.org/2001/03/reference.html*

[24] Johnson SC (1975) *Yacc: Yet Another Compiler Compiler*, Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, New Jersey

[25] Karp PD, Chaudhri V, Thomere J (1999) *XOL: An XML-Based Ontology Exchange Language*. Version 0.3. Technical Report. *http://www.ai.sri.com/~pkarp/xol/xol.html*

[26] Kifer M, Lausen G, Wu J (1995) *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM 42(4):741–843

[27] Klein M (2001) *Combining and relating ontologies: an analysis of problems and solutions*. In: Gómez-Pérez A, Grüninger M, Stuckenschmidt H, Uschold M (eds) IJCAI2001 Workshop on Ontologies and Information Sharing, Seattle, Washington

[28] Knublauch H (2003) *Editing Semantic Web Content with Protégé: the OWL Plugin*. 6th Protégé workshop. Manchester, United Kingdom

[29] Lassila O, Swick R (1999) *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. *http://www.w3.org/TR/REC-rdf-syntax/*

[30] Lenat DB, Guha RV (1990) *Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Boston, Massachusetts
[31] Lesk ME (1975) *Lex - A Lexical Analyzer Generator*, Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, New Jersey
[32] Luke S, Heflin J (2000) *SHOE 1.01. Proposed Specification*. Technical Report. Parallel Understanding Systems Group. Department of Computer Science. University of Maryland. *http://www.cs.umd.edu/projects/plus/SHOE/spec1.01.htm*
[33] MacGregor R (2001) *Inside the LOOM classifier*. SIGART Bulletin 2(3):88–92
[34] Maedche A, Motik B, Stojanovic L, Studer R, Volz R (2003) *Ontologies for Enterprise Knowledge Management*. IEEE Intelligent Systems 18(2):26–33
[35] Morris CW (1938) *Foundations of the theory of signs.* In: Neurath O, Carnap R, Morris CW (eds) International encyclopedia of unified science. Chicago University Press [reprinted in C W Morris 1971 *Writings on the theory of signs*. Mouton, The Hague]
[36] Motta E (1999) *Reusable Components for Knowledge Modelling: Principles and Case Studies in Parametric Design*. IOS Press. Amsterdam, The Netherlands
[37] Noy NF, Fergerson RW, Musen MA (2000) *The knowledge model of Protege-2000: Combining interoperability and flexibility*. In: Dieng R, Corby O (eds) 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00). Juan-Les-Pins, France. (Lecture Notes in Artificial Intelligence LNAI 1937) Springer-Verlag, Berlin, Germany, pp 17–32
[38] Schreiber G, Akkermans H, Anjewierden A, de Hoog R, Shadbolt N, van de Velde W, Wielinga B (1999) *Knowledge engineering and management. The CommonKADS Methodology*. MIT press, Cambridge, Massachusetts
[39] Studer R, Benjamins VR, Fensel D (1998) *Knowledge Engineering: Principles and Methods*. IEEE Transactions on Data and Knowledge Engineering 25(1-2):161–197
[40] Sure Y, Staab S, Angele J (2002) *OntoEdit: Guiding Ontology Development by Methodology and Inferencing*. In: Meersman R, Tari Z (eds) Confederated International Conferences CoopIS, DOA and ODBASE 2002, University of California, Irvine. (Lecture Notes in Computer Science LNCS 2519) Springer-Verlag, Berlin, Germany, pp 1205–1222
[41] Swartout B, Ramesh P, Knight K, Russ T (1997) *Toward Distributed Use of Large-Scale Ontologies*. In: Farquhar A, Gruninger M, Gómez-Pérez A, Uschold M, van der Vet P (eds) AAAI'97 Spring Symposium on Ontological Engineering. Stanford University, California, pp 138–148

Internet session: Ontolingua Server

http://ontolingua.stanford.edu/

The Ontolingua Server [Farquhar et al., 1997] was the first ontology tool created. It appeared in the mid-1990s, and was built to ease the development of Ontolingua ontologies with a form-based Web interface. Initially the main application inside the Ontolingua Server was the ontology editor. Then other systems were included in the environment, such as a Webster, an equation solver, an OKBC server, the ontology merge tool Chimaera, etc.

The Ontolingua language [Gruber, 1992] was created on top of KIF [Genesereth and Fikes, 1992] as a frame-based syntax for this language. KIF, and consequently Ontolingua, was aimed to be used as an interchange language between different knowledge representation systems, and for this reason it had a large expressiveness and was easily extensible. However, from a pragmatic point of view it failed to achieve its goal due to the many different possibilities available for the representation of the same piece of knowledge, what made it difficult to use it for interoperability.

In this session, we will focus on the Ontolingua Server ontology editor and its functions for transforming ontologies to different languages, including KIF and LOOM.

Interaction:

Enter in the Ontolingua Server web site and click on the button “Log in anonymously”. Then click on the link “Ontology Editor”, which opens the Ontolingua ontology editor in the window. As we have entered as an anonymous user we will not be able to perform all the operations that would be available if we registered nor will we be able to see and edit all the ontologies available in the library, but it is not necessary to have those for our purposes. Once in the first screen of the ontology editor we can proceed to click on the “Start Ontology Editor” link, which will show the main user interface of the application.

The interface shows the current ontology library stored in the server, with the loaded and unloaded ontologies. We will select one of them for our purpose: the Enterprise Ontology [Uschold et al, 1998][9], which defines a collection of terms and definitions relevant to business enterprises. The classes, relations, functions, axioms, and instances (individuals) of the ontology can be browsed from the main user interface, using the links provided in the main description page. We encourage our readers to take a look at it to become familiar with the ontology.

After having been browsing the ontology for some time, we will analyse the definition of one of the classes of the ontology: *Activity*. This can be done by selecting this term from any of the description web pages of the ontology. The description page of this class contains information about the class, such as its subclasses and superclasses, the relations of which it is a domain and range, the natural language description, the axioms where this class is involved, etc.

Now we will see the source code for this class in the Ontolingua language. If we go to the top frame in the page we can see a set of buttons and drop-down list. We select Download Translation from the drop-down list close to the button named Ontology. Once selected, we press that button. This will show another page with some more options. We will select the OKBC friendly option and then click in the Do It button. This will open in our browser a file with extension .gf that can be viewed with any text editor. That file contains the source code of the Ontolingua ontology. We can look now for the term Activity by looking for the expression “(define-okbc-frame Activity”. Once that we have found it we have the Ontolingua code for the class, which is as follows:

```
(define-okbc-frame Activity
  :frame-type :class
  :direct-superclasses (Activity-Or-Spec)
  :direct-types (Class Primitive)
  :own-slots ((Arity 1))
  :primitive-p common-lisp:nil
  :template-slots ((Actual-Activity-Interval) (Actual-Pre-Condition)
                   (Actual-Effect) (Activity-Status))
  :template-facets
  ((Actual-Activity-Interval (Value-Type Time-Range)
    (Cardinality 1) (Minimum-Cardinality 0))
   (Actual-Pre-Condition (Value-Type Pre-Condition)(Minimum-Cardinality 1))
   (Actual-Effect (Value-Type Effect) (Minimum-Cardinality 1))
   (Activity-Status (Value-Type Activity-State)(Minimum-Cardinality 1)))
  :sentences
   ((=> (Activity-Or-Spec ?x) (and (or (Activity ?x) (Activity-Spec ?x))(Eo-Entity ?x)))
    (=> (and (or (Activity ?x) (Activity-Spec ?x)) (Eo-Entity ?x))(Activity-Or-Spec ?x))
    (<= (Activity-Or-Spec ?x) (and (or (Activity ?x) (Activity-Spec ?x))(Eo-Entity ?x)))
    (<=> (Activity-Or-Spec ?x)(and (or (Activity ?x) (Activity-Spec ?x))(Eo-Entity ?x)))
    (=> (Activity-Or-Spec ?x)(or (Activity ?x) (Activity-Spec ?x)))))
```

This piece of code contains all the information that we already obtained in the user interface, except for that one that was obtained by inference. For instance, it does not provide information about which are the relations that have Activity as a range.

Let us now see how this is translated into a description logic language like LOOM. In the same page as before (the one where we selected the translation into OKBC), we select Loom as the output format of the ontology. Then we click again in Do It and we will obtain a file that can be as well opened with any text editor.

```
;;; Forward declaration for within ontology reference #%Activity.
;;; Forward declaration for #%Activity.
(loom:defconcept Activity :only-if-no-preexisting-definition-p common-lisp:t)

;;; --- In the definition of Class ACTIVITY
;;; Can't translate the following axioms:
;;;  (Template-Facet-Value Value-Type Activity-Status Activity Activity-State)
;;;  (Template-Facet-Value Minimum-Cardinality Activity-Status Activity 1)
;;;  (Template-Facet-Value Value-Type Actual-Effect Activity Effect)
;;;  (Template-Facet-Value Minimum-Cardinality Actual-Effect Activity 1)
;;;  (Template-Facet-Value Value-Type Actual-Pre-Condition Activity Pre-Condition)
;;;  (Template-Facet-Value Minimum-Cardinality Actual-Pre-Condition Activity 1)
;;;  (Template-Facet-Value Value-Type Actual-Activity-Interval Activity Time-Range)
;;;  (Template-Facet-Value Cardinality Actual-Activity-Interval Activity 1)
;;;  (Template-Facet-Value Minimum-Cardinality Actual-Activity-Interval Activity 0)
;;; Concept #%Activity
(loom:defconcept Activity :is-primitive Activity-Or-Spec :in-partition
 $activity-or-spec-partition-1$ :annotations
 ((Related-Axioms
   '(Template-Facet-Value Minimum-Cardinality Actual-Activity-Interval Activity 0))
  (Related-Axioms
   '(Template-Facet-Value Cardinality Actual-Activity-Interval Activity 1))
  (Related-Axioms
   '(Template-Facet-Value Value-Type Actual-Activity-Interval Activity Time-Range))
  (Related-Axioms
   '(Template-Facet-Value Minimum-Cardinality Actual-Pre-Condition Activity 1))
  (Related-Axioms
   '(Template-Facet-Value Value-Type Actual-Pre-Condition Activity Pre-Condition))
  (Related-Axioms
   '(Template-Facet-Value Minimum-Cardinality Actual-Effect Activity 1))
  (Related-Axioms
   '(Template-Facet-Value Value-Type Actual-Effect Activity Effect))
  (Related-Axioms
   '(Template-Facet-Value Minimum-Cardinality Activity-Status Activity 1))
  (Related-Axioms
   '(Template-Facet-Value Value-Type Activity-Status Activity Activity-State))
  (Documentation "Something done over a particular Time-Range. The following may pertain to
an Activity: * is performed by one or more Actual-Doer s; * is decomposed into more
detailed Sub-Activity s; * Can-Use-Resource s; * An Actor may Hold-Authority to perform it;
* there may be an Activity-Owner; * has a measured efficiency. ")))
```

We can perform the following comparison>

- All the information has not been transformed correctly into Loom. For instance, the Related-Axioms bits in Loom are only axioms that can be used to describe further a component in Loom, but are not used to create the ontology model in the language, so they have no effect in the semantics of the model.
- The Loom code contains a first sentence that defines the concept name in advance, so as to avoid problems when using the class Activity in the definition of other classes and relations, in case that it has not been defined yet. This would not be necessary if an API had been used that solved that problem.
- In the Ontolingua code the documentation of the class did not appear (due to a problem in the translation system) while in Loom it appears correctly.
- The general axioms that were represented in Ontolingua (in the :sentences clause) have not been translated to Loom.

Consequently, from this quick inspection of just one class we can conclude that the quality of the translation systems in the Ontolingua Server is not very good, and the main problem that we face is mainly due to the fact that we do not know in advance which are the design decisions that have been taken in the development of the translation system, since it has not been specified declaratively. Hence we do not know which pieces of knowledge are lost, how each type of sentence is transformed, etc.

Case study. Creating an ontology translation system between OWL and WSML

You are now responsible of developing an ontology translation system between two ontology languages that have recently appeared. On the one hand we have OWL, which was proposed as a W3C Recommendation on February 2004. This language is mainly based on the description logic paradigm, and has three different layers: OWL Lite, OWL DL and OWL Full. On the other hand we have WSML, which is being developed in the context of the WSMO[10] initiative for the description of Semantic Web Services. As a consequence of being in a development phase not all the aspects of the language are stable and may change in the following months.

The main references to be used for our work are the following:

- OWL reference: http://www.w3.org/TR/owl-ref/

- WSML reference: http://www.wsmo.org/TR/d16/d16.1/v0.21/ (current version 0.21)

Besides, we will use two APIs, each one for each of the languages:

- Jena API for OWL ontologies: http://jena.sourceforge.net/

- WSMO4J API for WSML ontologies: http://wsmo4j.sourceforge.net/

As proposed in the method presented in this paper, the first activity will consist in the performance of a **feasibility study** of the translation system to be developed. This activity consists of 4 tasks, as described in figure 7, where we determine the scope of the translation system, the expected outcomes, the context where it will be used.

To help with this feasibility study, in [Corcho, 2005] we can find forms that can be used to perform these assessments. However, here we will not need to go into so much detail, since we know that it is a strong requirement to have such a translator (currently there are no translators available and hence when we want to describe Semantic Web Services according to OWL ontologies we have to duplicate those ontologies in WSML). However, since the context may change in the future, we will require here the analysis of existing translators between both systems, an overview of the main requirements for such a system, restrictions for it, etc.

Once the feasibility study has been performed, and assuming that the result was positive, we proceed to the **analysis of the source and target formats**. In the description of WSML and in some other WSMO deliverables there are formal comparisons between OWL and WSML, determining the similarities and differences between both languages. Hence this information can be used as an input to this activity. We propose to "transform" this formal comparison into a semi-formal comparison using the approach presented in [Gómez-Pérez et al., 2003]. As a result of this comparison we will obtain mappings between the different components in each of the languages, and we will know which components or part of the components cannot be expressed in each of the formats. Finally, we have to propose a test plan, and for this we propose the use of existing ontologies, which can be found in existing libraries like the DAML or Protégé ontology libraries for OWL ontologies, or the WSMO library for WSML ontologies, or using search and peer-to-peer tools like Swoogle or Oyster respectively.

Once the analysis has been made, we proceed to the **design of the ontology translation system**. Here we have to identify the transformations to be made at the different layers described in this paper: lexical, syntax, semantic and pragmatic. For the lexical and syntax

layers there will not be a huge need to study in depth the syntax of the two languages, since we propose the use of ontology APIs for them. However, it will require the study of those APIs, whose documentation can be found in the URLs previously pointed out. In order not to make this exercise too complex, we propose to study only the transformation of OWL classes into WSML classes, though this can be extended if we want to make a more exhaustive exercise.

Finally, once that the design has been made (although we may have to come back to it later) we move towards the **implementation of the ontology translation system**. In section 3 we have proposed the use of different languages for this implementation (ODELex, ODESyntax and ODESem), whose description can be found in [Corcho, 2005]. If you are familiar with the languages lex and yacc, or similar ones, you will find it easy to use them. However, if you are unfamiliar with them, and given the fact that the design has been done according to the layers proposed before, then you can proceed directly with the implementation of the translation system without using them. We will only use some of the advantages of using declarative code, but since we have carefully designed the translation system according to layers, the implementation will be easy, since at each step we will perform different sets of transformations.

Finally, we will test the resulting implementation (transformation of classes from OWL to WSML) and will go back to any of the previous activities in case that something does not comply with our test cases.

The resulting code can be then made available as a Web service and can be used as an ooMediator inside WSML descriptions, so as to allow the import of OWL ontologies into WSML ontologies.

Useful URLs

Toolset for transformations between XML Schema, RDF and OWL: http://rhizomik.upf.edu/redefer/

EXMO and Transmorpher: http://www.inrialpes.fr/exmo/

Benchmarking of interoperability between ontology tools: http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

OntoMorph: http://www.isi.edu/~hans/ontomorph/presentation/ontomorph.html

KnowledgeWeb network of excellence: http://knowledgeweb.semanticweb.org

OntoWeb thematic network: http://www.ontoweb.org/. Special attention should be put to deliverable D1.3 about ontology tools.

Further readings

Borgida A (1996) *On the relative expressiveness of description logics and predicate logics*. Artificial Intelligence 82(1-2):353–367

Chalupsky H (2000) *OntoMorph: a translation system for symbolic knowledge*. In: Cohn AG, Giunchiglia F, Selman B (eds) 7th International Conference on Knowledge Representation and Reasoning (KR'00). Breckenridge, Colorado. Morgan Kaufmann Publishers, San Francisco, California, pp 471–482

Corcho O (2005) *A layered declarative approach to ontology translation with knowledge preservation*. Frontiers in Artificial Intelligence and its Applications. Dissertations in Artificial Intelligence. IOS Press

Euzenat J, Tardif L (2001) *XML transformation flow processing*. Markup languages: theory and practice 3(3):285–311

Gómez-Pérez A, Fernández-López M, Corcho O (2003) *Ontological Engineering: with examples from the areas of knowledge management, e-commerce and the Semantic Web*, Springer-Verlag, New York.

Omelayenko B, Klein MCA (eds). Knowledge Transformation for the Semantic Web. Frontiers in Artificial Intelligence and Applications Vol. 95 IOS Press 2003

Sure Y, Corcho O, Angele J (eds). Proceedings of the ISWC2003 workshop on Evaluation of Ontology Tools (EON2003). Available at http://km.aifb.uni-karlsruhe.de/ws/eon2003

Possible paper titles / essays

Incremental translation between ontology languages and/or tools with ontology versioning support

Formalism-based library of reusable translation functions between languages and/or tools

Dynamic ontology translation systems between languages and/or tools

Use of formal language comparison models in the development of ontology translation systems.